PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 13/00, 13/24

A1

(11) International Publication Number: WO 97/10548

(43) International Publication Date: 20 March 1997 (20.03.97)

(21) International Application Number: PCT/US96/14568

(22) International Filing Date: 11 September 1996 (11.09.96)

(30) Priority Data:

60/003,561	11 September 1995 (11.09.95)	US
Not furnished	11 September 1996 (11.09.96)	US

(71) Applicant: MCSB TECHNOLOGY CORPORATION [US/US]; 4330 Golf Terrace, Eau Claire, WI 54701 (US).

(72) Inventors: SHEETS, Kitrick; 2179 Aebly Road, Chippewa Falls, WI 54729 (US). THOMPSON, Keith; 3268 Oxford Drive, Woodbury, MN 55125 (US).

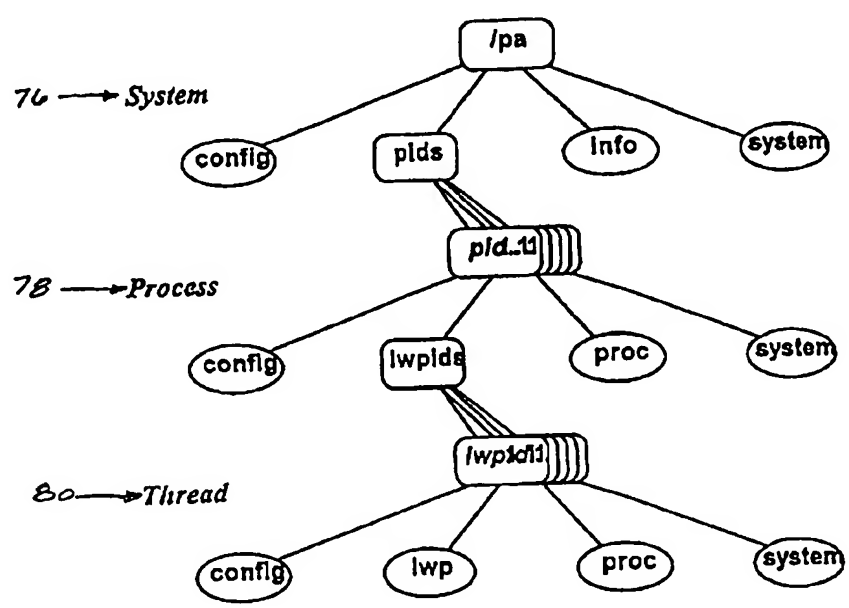
(74) Agents: STANGA, Paul, W. et al.; Patterson & Keough, P.A., 1200 Rand Tower, 527 Marquette Avenue South, Minneapolis, MN 55402 (US).

(81) Designated States: AU, CN, JP, KR, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published

With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: PERFORMANCE ASSISTANT FILE SYSTEM (PAFS) METHOD AND APPARATUS



(57) Abstract

A performance enhancement system including a software implemented in a Performance Assistant File System (PAFS, 52) to enable an efficient and economical performance evaluation and tuning of data servers (58, 60, 62, 64) and networks is disclosed. The PAFS of the present invention is a subcomponent of a performance assistant (PA, 72, 74) architecture which includes a set of powerful software tools (54) that help the user/system administrator tune a computer system to yield maximum performance. Specifically, the present invention enables the user to diagnose and tune operating systems without the need for utilizing laboratory conditions and tools such that the full potential of the processors architecture could be realized.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	GB	United Kingdom	MW	Malawi
AT	Austria	GE	Georgia	MX	Mexico
AU	Australia	GN	Guinea	NE	Niger
BB	Barbados	GR	Greece	NL	Netherlands
BE	Belgium	HU	Hungary	NO	Norway
BF	Burkina Faso	IE	Ireland	NZ	New Zealand
BG	Bulgaria	IT	Italy	PL	Poland
BJ	Benin	JP	Japan	PT	Portugal
BR	Brazil	KE	Kenya	RO	Romania
BY	Belarus	KG	Kyrgystan	RU	Russian Federation
CA	Canada	KP	Democratic People's Republic of Korea	SD	Sudan
CF	Central African Republic			SE	Sweden
CG	Congo	KR	Republic of Korea	SG	Singapore
CH	Switzerland	KZ	Kazakhstan	SI	Slovenia
CI	Côte d'Ivoire	LI	Liechtenstein	SK	Slovakia
CM	Cameroon	LK	Sri Lanka	SN	Senegal
CN	China	LR	Liberia	SZ	Swaziland
CS	Czechoslovakia	LT	Lithuania	TD	Chad
CZ	Czech Republic	LU	Luxembourg	TG	Togo
DE	Germany	LV	Latvia	TJ	Tajikistan
DK	Denmark	MC	Monaco	TT	Trinidad and Tobago
EE	Estonia	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	UG	Uganda
FI	Finland	ML	Mali	US	United States of America
FR	France	MN	Mongolia	UZ	Uzbekistan
GA	Gabon	MR	Mauritania	VN	Viet Nam

PERFORMANCE ASSISTANT FILE SYSTEM (PAFS) METHOD AND APPARATUS

RELATED APPLICATIONS

This application is a formal application of a part of the Provisional Application filed on September 11, 1995 and assigned Serial Number 60/003,561.

FIELD OF THE INVENTION

The invention relates generally to capacity and workload optimization of data servers and workstations. More specifically the present invention pertains to performance enhancement using a performance assistant file system (PAFS) to enable a correlation between performance data collected from a hardware system and the application running on the system.

DESCRIPTION OF RELATED ART

Data servers and workstations are well known in the art. The computing industry has moved toward distributive computing networks comprised of heterogeneous workstations, connected together in an open system network. As a result, there has been an ever increasing demand for more powerful and faster data servers to service both the distributive data management and processing requirements of users within the network. In response to this ever increasing demand, multiprocessing data servers, such as described in U.S. Patent No.'s 5,355,453, and 5,163,131 to Row, et al., have been developed.

Unlike traditional mainframe computer systems and subsystems for mainframe computers which generally operate in a captive, homogenous environment, network data servers must be capable of responding to resource requests from a wide variety of users in a heterogeneous network environment. The data server must respond to these requests in an efficient, yet distributed, manner without any type of central control from a mainframe computer. As such, the problems and demands imposed on

the design of a system architecture for a network data server are significantly different from a mainframe computer systems or subsystem.

While the design of existing data servers has been sufficient to accommodate the demands of most users in a network computer environment, it would be advantageous to provide a performance assistant (PA) architecture for a data server to enable monitoring of applications or systems without the need for the applications to be compiled in any special way to effect monitoring. It would further be advantageous to provide a performance assistant file system (PAFS) to enhance performance, scalability, robustness and maintainability required for the ever-increasing demands of network client-server applications.

BACKGROUND OF THE INVENTION

Performance information about a computing system is critical in evaluating, diagnosing and developing enhancement features and options. Prior art performance evaluation and enhancement systems interfered with the application system or hardware being monitored thus making it impractical to work with an active application running on a system/hardware.

Further, the present state of the art is such that evaluation and diagnosis to optimize performance is rather complex and requires intensive labor and various tools. In order to get maximum performance from a software implemented in a computer or an application executing on any system, it is important to understand how the software code interacts with the underlying hardware components. A successful performance evaluation should, at a minimum, be able to examine how each of the hardware components interact with the software implemented therein.

Techniques for optimizing performance are now well known, but as indicated hereinabove, the required intensive labor and the associated tools make it impractical to routinely and efficiently use these techniques in an increasingly complex environment of modern servers and

workstations.

To simplify and significantly enhance the process of system performance, evaluation and tuning there is a need to integrate software in a performance assistant (PA) architecture. Further, it is required that such software must provide a set of compatible tools. The software and associated tools must further enable a generic interface for the collection of performance data and the correlation of this data to executing tasks within a system hardware.

Accordingly, any optimization system and software, dedicated to diagnose and tune data servers and workstations, must employ efficient and reliable tools. Thus, there is a need for an interactive and diagnostic system, which is compatible with the evolving complex environment of servers and workstations, to enable a real-time, comprehensive and automatic optimization of task/workload allocation for data servers and workstations.

SUMMARY OF THE INVENTION

The Performance Assistant ™ File System (PAFS) is a unique method and apparatus for accessing performance information about a computing system. The PAFS makes performance information such as bus activity, processors performance, I/O performance and operating system information appear to be stored in a special file system that is mounted like any other Unix® file system. Because PAFS appears to be a file system, a system programmer can use ordinary file handling subroutine calls to access the specification and data. The PAFS is preferably a hierarchical file system that can collect performance information for any individual processor or light weight process operating in the computing system.

The PA includes a set of specialized software tools that help the system administrator or the programmer to tune the computer system and enhance performance. In order to maximize its effectiveness, the PA specifically targets systems, system elements and processes. For example,

effective use of caches, memory bottlenecks, I/O bottlenecks, lock contention bottlenecks and use or overuse of the system bus are among the many targets on which the PA software may be targeted.

Unlike most file systems, PAFS has no permanent store. It comes into existence when mounted and disappears when unmounted. The files and directories that appear under the PAFS mountpoint reflect an access method rather than a real entity. Further, directories appear and disappear automatically as processes come and go on the running system thus reflecting the current state of the system.

The PAFS is a central connection point of the performance assistant (PA) architecture. It acts as the clearing house for all performance data collected from the hardware while supplying the critical correlation between that data and application running on the system. The PA includes a set of powerful software tools that help the user to tune the computer system to run at peak performance.

One of the most important features of the PA software is that, unlike prior art systems, it enables monitoring of applications/systems without the need for the applications/systems to be compiled in any special way to effect the monitoring. In fact, the PA software remains latent and passive such that the application being monitored is blind to the existence of the PA software.

With these and several other advantages the PAFS, which is a significant element of the PA, provides a targeted performance evaluation and enhancement tool for applications/systems as well as a particular hardware in which the application/system is implemented. It will be appreciated that the methods and apparatus of the present invention are advancements over prior art methods and apparatus. Other features and advantages of the present invention will become apparent upon examination of the following description and drawings dealing with several specific embodiments thereof.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a prior art block diagram of a symmetrical multiprocessing (SMP) computer system.

Figure 2 is a functional view of a superscalar processor.

Figure 3 is a block diagram of a pipelined processor architecture.

Figure 4 is a structure depicting the performance assistant environment.

Figure 5 is a conceptual view of a performance assistant file system (PAFS).

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention provides a PAFS which is a part of a PA system. Primarily, the PA environment includes a set of tools, utilities and libraries designed to allow users to achieve maximum performance from software running on their server or workstation and in turn help maximize their investments in the underlying hardware. The components included in this architecture range from tools designed to provide developers in-depth information about the execution characteristics of their application, to operating system modules designed to automatically optimize the current application workload of a server in real-time.

The present invention is advantageously applicable to data servers and workstations. In order to provide a basis for understanding the present invention and its many advantageous features, prior art computer systems and relevant processor architecture are discussed hereinbelow.

Figure 1 is a standard symmetric multiprocessor (SMP) 10 which is typical of most server systems available today. Generally such systems consist a number of processors which reside on a shared system bus. In Figure 1, CPU 12 depicts a plurality of such processors. CPU 12 includes a first level (L1) 14 cache system. Further, each processor has a second level (L2) 16 cache memory and communicates with other processing elements

through a single shared global memory. Memory 18, Network 20 and disk I/O 22 share an interconnect path 23 with CPU 12. Similarly data path 23' interconnects CPU 12, memory 18, network 20 and disk I/O 22. Protocol and consistency is maintained by special algorithms designed to monitor the system bus and maintain consistency among processor caches, such as L2 cache 16 and memory.

Much work has been done over the last several years to enhance the performance of microprocessor architectures. This has led to various architectural configurations in SMP computer systems. Techniques such as parallel execution, pipelining and speculative processing are becoming common in most microprocessors built today.

Parallel execution utilizes multiple functional units into a chip and installs multiple operations on a single processor cycle. A processor with such capability is called "superscalar". Figure 2 is a simplified block diagram of a superscalar processor architecture 30. The functional aspects include instruction stream 32, functional units 34 and system interface 36.

A typical prior art example of pipelining is shown in Figure 3. Pipelining techniques are implemented to make up for performance losses which occur in a processor when instructions take multiple cycles to execute. In a pipelined architecture instructions move through the processor one stage at a time until the entire instruction has been executed. Figure 3 provides a functional view of such an architecture. Instruction stream 40 passes through pipeline stages 42 to system 44 and the cycle is repeated as apparent. This technique does not reduce the amount of time required to execute a given instruction. However, it permits execution time of the instruction to be overlapped thereby reducing overall circuitry occupancy for a given period time.

Although pipelining enhances the performance of some codes, there are limitations to this technique. A major problem occurs when the processor branches to a new piece of code and experience pipeline stall. Techniques such as branch prediction and speculative execution have been developed to examine the execution path of an application and predict

what code segment will be needed in the immediate future execution.

In the context of the present invention software plays an important role in the optimization of data servers and workstations. On any server it is important that applications run as efficiently as possible. Generally, efficiency is dependent on utilization of processor resource and cache. As indicated in supra, modern processors are equipped with multiple functional units. These functional units and their associated pipelines can be utilized to allow the processor to deploy all of its available resources on the application. Further, the interaction of the application with the processor's cache architecture is also crucial to the performance and overall system efficiency.

Furthermore, interprocessor communication (IPC) is becoming a common place occurrence in servers and network systems. This becomes even more common as applications are designed to take advantage of lightweight thread interfaces, available in most modern operating systems, to break application tasks into smaller pieces to communicate or consolidate data. When such code is executed in multiprocessor systems the need to avoid cache thrashing conditions becomes critical.

Moreover, system performance is dependent upon operating system software. The operating system controls the utilization of hardware resources and needs to be efficient in managing this allocation function. In multiprocessor operating system, for example, internal data structures are protected from simultaneous update through the use of a variety locking algorithms. At the base of these locking routines are hardware primitives which support atomic memory update transactions. To protect structures which require frequent and fast updates locking, routines are used which loop on the lock variable until it is acquired exclusively (spin locks). Sleep locks or semaphores are used when a lock is not available. Spin locks are used for fine grained locking and semaphores are used for coarse grained locking.

Yet another performance optimization which is often performed on operating systems is hand coding or algorithmic optimization of the most

commonly executed code sections.

To examine system performance related activity at the hardware level, in circuit emulation (ICE) devices or logic analyzers are sometimes used. These devices capture signals on various buses within the system and store and/or display the results of that trace. Any information which appears on a processor or system bus can be examined with these devices. Although these systems are useful for monitoring the low level details of the system hardware, the devices are generally expensive.

While each of the tools in the prior art discussed in supra are useful for the specific tasks for which they are designed, there are circumstances where some of these tools become impractical or impossible to use. For example, current tools are less than optimal for evaluating and tuning the performance of applications/systems as a whole. This is primarily due to intrusiveness, coarse level of coverage, disjoint tool set and limited end-user use. Probably the most limiting factor on current hardware and software performance tuning techniques is the limitation by the end-user. Most tuning devices are designed for use in development or laboratory environments.

Referring now to Figure 4 a structure of the Performance Assistant (PA) architecture 50, in which the present invention is implemented is shown. Specifically, performance assistant file system (PAFS) 52 is shown as the central connection point of PA architecture 50. PAFS 52 acts as a clearing house for all performance data collected from the hardware while supplying the critical correlation between that data and applications running on the system. PAFS 52 is connected to Autopilot 54. Base operating system hooks 56 includes a two-way communication with Autopilot 54 and PAFS 52. Further, Engine Performance Monitors (EPM) 58 and EPM 62 are in a two-way data communication with PAFS 52. Bus Performance Monitor (BPM) 62 provides system bus activity processor feedback and is also in a two-way communication with PAFS 52. PAFS 52 enables additional drivers 64 to be added depending on the capabilities provided by the underlying hardware platform. PAFS 52 is also in a two-

way communication with PAFS access library 66. PAFS access library 66 enables and provides a two-way communication with AutoPilot control 68, PAFS control 70, PA configuration 72 and PARun 74 and other controls 76.

Figure 5 shows a conceptual view (tree) of PAFS 52. As shown in Figure 4, PAFS 52 is the central connection point of the PA architecture 50. Referring now to Figure 5 in more detail, at the root of the tree, PAFS 52 maintains system 76 to provide information relating to the overall architecture as a whole. As the user moves down the tree, the granularity of the data becomes more fine. The root directory comprises for each process 78, currently active in system 76, one directory. Further, within each process 78 there is a directory for each active thread 80. Using this tree structure, PAFS 52 maintains a consistent, accurate accounting of the process hierarchy of the active system.

Referring now to Figure 4 in more detail, Pentium Engine Performance Monitors 58 and 60 allow key performance related statistics to be gathered. For example a partial list of the performance data to be collected includes cache hit rates, branch statistics, pipeline utilization, functional unit utilization, instruction counts, interrupt counts and memory management unit statistics. The present invention incorporates this continuous feedback to determine the current behavior of the code and utilizes the information to tune an application such that the full potential of the processor architecture may be reached.

Similarly, base operating system hooks 56 are used to notify PAFS 52 when important events occur within the base operating system. These events include thread creation, thread exit and thread context switches. Receiving notification at these times allows PAFS 52 to precisely track the execution history of each thread in the system and provide meaningful performance data for each running application.

Referring now to Figure 4 in yet more detail, major components which together form the PA 50 environment are shown. Primarily these components may be conveniently divided into a data collection interface,

PAFS 52 user interface and PA 50 tools. These components are discussed in detail hereinbelow.

The basis for all capabilities provided by the PA 50 environment is its low-level data collection interface. This foundation component is made up of modules which are responsible for the configuration of performance related hardware interfaces and the maintenance of performance data collected from these hardware components. The data collection interface of PA 540 includes PAFS 52. As discussed in supra, PAFS 52 is the central connection point of PA 50 architecture and acts as a clearing house for all performance data collected from the hardware and simultaneously supplies the critical correlation between that data and applications running on the system. As shown in Figure 5, data collected by drivers is forwarded to PAFS 52 which is thereby made available for use and access by the other components in the system.

As discussed herein supra, PAFS 52 is able to correlate the data it receives from underlying drivers through its use of a few strategically placed hooks in the base operating system. These hooks are key to PAFS's 52 ability to provide meaningful data to users. The main function of the hooks is to notify PAFS 52 when important events occur within the base operating system. These events include thread creation, thread exit, and thread context switches. Receiving notification at these times allows PAFS 52 to precisely track the execution history of each thread in the system and provide meaningful performance data for each running application. Since the structure of a system's processes and threads is hierarchical in nature, it is natural that the data repository for the Performance Assistance architecture take the form of a file system; but this is not absolutely necessary.

Referring now to Figure 5, at the root of the tree PAFS 52 maintains information relative to the system as a whole. Within each process directory there is a directory for each active thread 88 within that process. At each level in the tree there is a configuration file which is used to configure a counter to collect data of interest. The types of counters

available depend upon the capabilities of the underlying hardware and its associated PAFS 52 driver. Also at each level are data files which contain the collected performance data. PAFS 52 maintains data files for counters configured at that level as well as counters configured at higher levels in the process hierarchy. For example, if the user is interested in measuring the number of instructions executed in the system over some interval, that configuration would be placed in the *config* file in PAFS's 52 root directory. The data for this configuration would appear in the *data* file in the same directory. In addition, each process and thread directory would contain a *system data* file which would contain that process or thread's contribution to the system total. All of this is maintained automatically by PAFS 52 in the normal course of its data collection process.

One point to note about PAFS 52 is that all of its internal counters are maintained as 64 bit entities. This is necessary since the type of data which is being collected by PAFS 52 can quickly run out of space if 32 bit values were used. For example, assume that the user is interested in counting the number of processor cycles over some execution interval for a 200MHz processor. If 32 bit values were to be used, the counter would overflow in about 16 seconds. Moving to 64 bits provides the capacity to count the same value for over 1700 years before overflowing the counter.

As described hereinabove, PAFS 52 provides a generic interface for the collection of performance data and the correlation of that data to executing tasks within the system. However, special hardware are used to collect the data and provide pertinent information. This is controlled by special hardware drivers which interface with PAFS 52 and control underlying hardware performance collection agents. This section focuses on these drivers and how they cooperate with PAFS to provide in-depth performance data.

Generally, computer systems from different manufacturers have distinct architectures and performance characteristics. As a result, each of these machines require slightly different hardware to monitor their performance. To assist in the support of these novel disparate interfaces,

PAFS 52 provides an extensible driver interface. This driver interface allows the details of the control of these performance collection agents to be hidden within modules specifically designed to handle this task. In addition, the PAFS 52 driver interface allows support of additional performance data collection agents by simply plugging in an additional driver to control that agent. The concept is similar to that used to support a new peripheral device in a standard operating system. The following sections described two such drivers currently supported within the PAFS 52 design.

Hidden within most modern microprocessors are registers which allow key performance related statistics to be gathered. Included in this list are items such as: cache hit rates; branch statistics; pipeline utilization; functional unit utilization; instruction counts; interrupt counts and memory management unit statistics. These registers provide continuous feedback on the current behavior of code executing on the processor. Having access to this type of processor performance information gives the truest indication of the exact utilization the current code sequence is making of the available processor resources. This type of data is indispensable in tuning an application to reach the full potential of the processor architecture. EPM 58 and 60 include, inter alia, the registers listed hereinabove.

Although the Pentium's resident performance monitoring capabilities provide powerful features in support of low level processor performance feedback, it does not provide information about events which occur external to the CPU. Having the ability to monitor events which occur external to the CPU can be a tremendous aid in the optimization of system performance. For this reason, some systems manufactured today provide special hardware which monitors system bus activity and allows interactions between system components to be measured. One example of this is the Bus Performance Monitor developed by Chen Systems for the CHEN-1000 server.

BPM 62 is a C-bus II based peripheral card which can be used to

monitor transactions which occur on the system bus. Through the use of the set of control registers provided, the board can be configured to log the occurrence of any C-bus II transactions that may occur on the bus. In this way, bus traffic can be monitored to directly pinpoint which bus element(s) are responsible for system bus traffic during a certain time interval. In addition, it is possible to configure the board such that on only a certain element or transaction type of interest is being monitored. This card was designed to provide the types of in-depth monitoring support normally provided by In Circuit Emulation (ICE) devices.

In addition to straight bus transaction counting, BPM 62 can also be configured to monitor bus activity relative to memory address ranges. This feature extends the facility of the BPM 62 beyond simple bus transaction counting and general system load observation. Using this facility, it is possible to relate system bus activity to a specific code segment within an application.

Although PAFS 52 provides a flexible interface for the collection of performance data and the logical presentation of that data to the outside world, it can be programmatically cumbersome for the user to locate the appropriate files for the configuration of counters and extraction of associated data. For this reason, two interfaces are provided to simplify the user's interaction with PAFS. This first, called Performance Assistant Configuration Language (PACL), is a free-form language which allows users to specify counter configurations in a simple, standard form understood by all PA components. The second, the PAFS access library (for PAFSlib) 66 helps programmers take advantage of PAFS's 52 capabilities without concern for the underlying file system details.

As described hereinabove, PACL is a simple, free-form language which provides a consistent means of describing PAFS 52 counter configurations. The best way to describe the capabilities of PACL is through the use of a simple example. Let's assume that a user is interested in the number of read transactions ont he system bus attributable to some application. The following PACL statement would define the counter

configuration used to obtain that data:

```
"read transactions" : bpm
        mode  = read
        arb = cpu
        targ = any
        cycle = start
```

This example defines a counter named *read transactions* which uses the bpm driver to count the number of read transaction s from a CPU to any target element in the system. The *mode, arb, targ* and *cycle* fields are defined by underlying bpm hardware design and configurable through the bpm driver. While valid field names and definitions are dependent upon the underlying driver being accessed, PACL provides a simple, consistent interface which can be used by applications that require access to the capabilities of PAFS and its supported hardware interfaces.

The PACL interface, like PAFS 52, supports the ability to plug in new interface definitions which support the capabilities of associated PAFS drivers. Support for new devices is created by supplying a simple library which defines the configuration variables for the hardware interface and valid values for each variable.

The PAFS access library (PAFSlib) 66 is provided to make configuring counters and accessing data stored in PAFS 52 easier for the programmer. PAFSLib abstracts away the details of the underlying PAFS 52 implementation and provides the user with an interface which is specifically designed for easy counter access. The following example illustrates the use of the PAFSlib 66 API by an application:

```
PAFS        *pp = pafs_start (NULL, 0);
pacl_t      *lp = pacl_str ("\"bus read\" :bpm mode-=read". 0);
paval_t     val;
pafs_putcfg_name(pp, lp->name, getpid(), PAFS_LWPANY, &lp->cfg);
```

```
/* do something interesting here, then... */
pafs_getval_name(pp, lp->name, PAFS_LWPANY, &val);

/* make use of val.pv_cnt and fields as needed, then... */
pacl_free (lp);
pafs_end(pp);
```

In this example, PAFSlib 66 is used to instrument a specific code segment within an application to count the number of read accesses across the bus. As can be seen in this example, PACL is being used to define the bpm configuration required to collect the requested data. PAFSlib 66 takes care of the translation of this configuration definition into a form which is understood by PAFS 52. The user need not be concerned with the underlying structure or implementation of PAFS 52 on that machine.

The previous sections have described interfaces provided to support the use of the PA 50 architecture to evaluate and tune the performance of applications under development. However, as was described earlier, it is important to have the ability to monitor and tune the performance of systems and applications in a production environment. It is also important to perform this evaluation in a non-intrusive way. To support this goal, PA 50 provides a pair of tools designed to allow users to take full advantage of the PA 50 architecture on their production systems without impacting the performance or functionality of their applications.

A general description of the function of each tool is given below. For simplicity, Unix command line examples are presented here. In some Unix or MS Windows environments, these tools may take a graphical form. However, the basic concept of the tools will be similar to that presented here. More detailed information on the function of Performance Assistant tools for various operating system environments can be found in the respective specifications for each tool.

To assist in the collection of performance data for applications which are currently executing within the system, PA 50 provides the

PAConfig 72 tool. This tools give the user the flexibility to specify the type of data which is to be collected for which executable entities within the system.

```
$ paconfig -p100 "\"bus read\" :bpm mode=read"
$ pagetval -e"bus read" -p100
"bus read": 26 (16325312)
```

As in the PAFSlib 66 programming example shown above, the user wants to see how many system bus accesses his application is making during a certain interval. Using the *paconfig* tool to attach to a running version of the program, the user can obtain similar data without the need to instrument his code. Accompany tools such a *pagetval* in this example are used to retrieve the data collected for the application.

While PAConfig 72 provides a very useful interface for collecting performance data for running applications, this tool lacks the ability to follow the execution path of a task from start to finish. For this reason, the PARun74 interface has been added. This tools allows a user to configure performance counters which will cover the entire execution life of an application. By specifying the application to run and the appropriate counter configuration in PACL form, the user can obtain in-depth performance data for any application of interest. The following example shows how the PARun 74 tool may be used on a Unix command line:

```
$ parun -d"\" reads \": epm mode=dmemr" find /m -exec ls -ld { } \;
"reads": 73430 (358223)
```

In this example, the user is interested in determining the number of reads from the application's data space during the lifetime of the specified *find* command. To obtain this data, the user specifies the use of the epm driver's *dmemr* (data memory read) capability which correlates to the function by the same name available in the Pentium processor's

performance monitor registers. From the results of this command, the user sees that about 20% of the processor cycles over this interval (73430/358223) were data reads.

While this sample is fairly trivial, it illustrates the ability of these user tools to take advantage of the programmatic and data collection mechanisms provided by the other Performance Assistant components.

Accordingly, PA 50 architecture and its subcomponent PAFS 52 provide a strong base for tuning applications. In the context of the present invention, PAFS 52 enables easy access to hardware interaction information by utilizing Bus performance monitor (BPM) 62 and event performance monitors (EPMs) 60 and 62. EPM60 and 62 include a set of registers built into Intel's Pentium processor that non-invasively monitors the pentium processor's activities.

While the preferred embodiments of the invention have been shown and described, it will be obvious to those skilled in the art that changes, variations and modifications may be made therein without departing from the invention in its broader aspects and, therefore, the aim in the appended claims is to cover such changes and modifications as fall within the scope and spirit of the invention.

WHAT IS CLAIMED IS:

1. A performance assistant file system (PAFS) forming a subsystem of a performance assistant (PA) architecture including software tools implemented in a monitoring interface of a hardware, for server systems and the like, the hardware-implemented PAFS comprising:

the hardware system and an application running on the system;

the PA architecture implemented in the hardware; and

the PAFS forming a common connection point of the PA architecture and performance data collected from the hardware tothereby supply a real-time correlation between said performance data and said application running on the hardware wherein said correlation is indicative of a performance efficiency of said server system.

2. The PAFS of claim 1 wherein said common connection includes:

a two-way communication with base operating system hooks;

a two way communication with an autopilot system;

a two way communication with a plurality of engine performance monitor processors;

a two-way communication with a bus performance monitor;

a two-way communication option with other drivers base on the capabilities of said the hardware; and

a two-way communication with a PAFS access library.

3. The PAFS according to claim 2 wherein said two-way communications with said PAFS access library includes an indirect two-way communications between said PAFS system and AutoPilot control, PAFS control, PA configuration, PARun and other controls.

4. The PAFS of claim 1 further comprising:

an architecture including a tree structure; and

said architecture maintained by the PAFS system to provide information relating to said architecture.

5. The PAFS of claim 4 wherein granularity of data becomes finer as data is traced down said tree.

6. The PAFS of claim 4 wherein a root directory comprises a directory for each active process wherein said process further includes a directory for each active thread.

7. The PAFS system of claim 6 wherein said tree structure provides an accurate accounting of said process hierarchy of said system.

8. A performance assistant file system (PAFS) forming a subsystem of a performance assistant (PA) architecture including software tools implemented in a monitoring interface of a hardware, for server systems and the like, the hardware-implemented PAFS comprising:

the hardware system and an application running on the system;

the PA architecture implemented in the hardware; and

the PAFS including:

a data collection interface;

a user interface; and

performance enhancement tools;

the PAFS forming a common connection point of the PA architecture and performance data collected from the hardware to thereby supply a real-time correlation between said performance data and said application running on the hardware wherein said correlation is indicative of a performance efficiency of said server system.

9. The PAFS of claim 8 wherein said data collection interface includes drivers for collecting data which is forwarded to the PAFS and made available for use by the hardware and said application.

10. The PAFS of claim 9 wherein said drivers in said interface is extensible and supports mapping of said drivers to new hardware collection agents.

11. The PAFS of claim 8 wherein said user interface includes a performance assistant configuration language and PAFS access library.

12. The PAFS of claim 8 wherein said performance enhancement tools include performance assistant configuration tools to assist in real-time collection of performance data for said application running on the hardware.

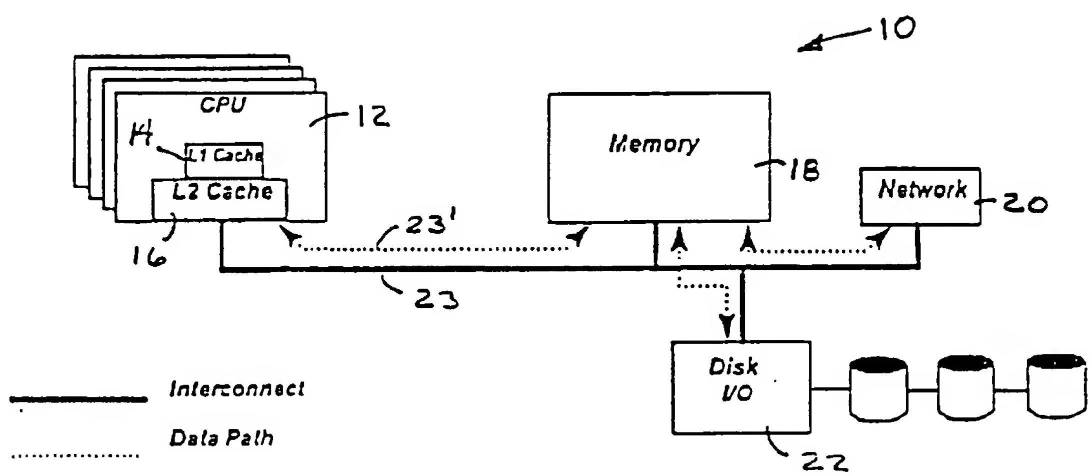

10
CPU
12
14
L1 Cache
L2 Cache
16
Memory
18
Network
20
23'
23
Disk
I/O
22
Interconnect
Data Path

Fig. 1

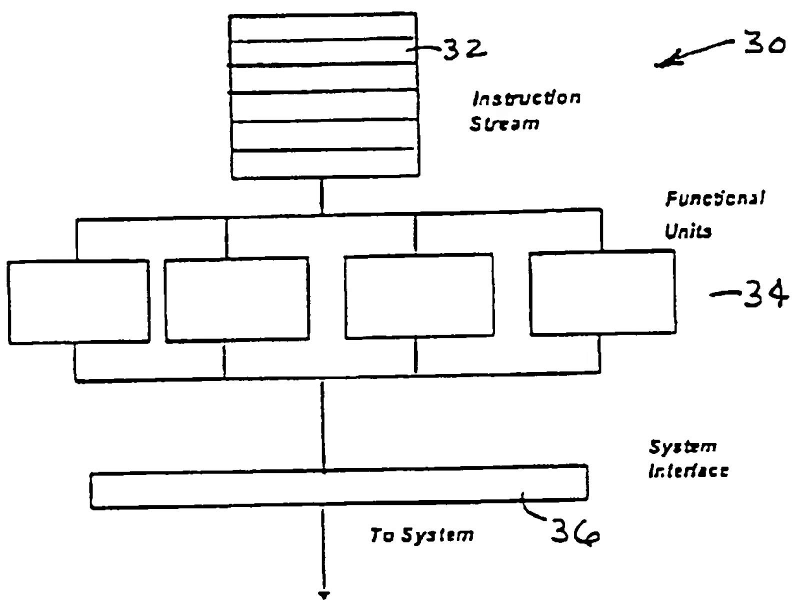

32
30
Instruction Stream
Functional Units
34
System Interface
To System
36

Fig. 2

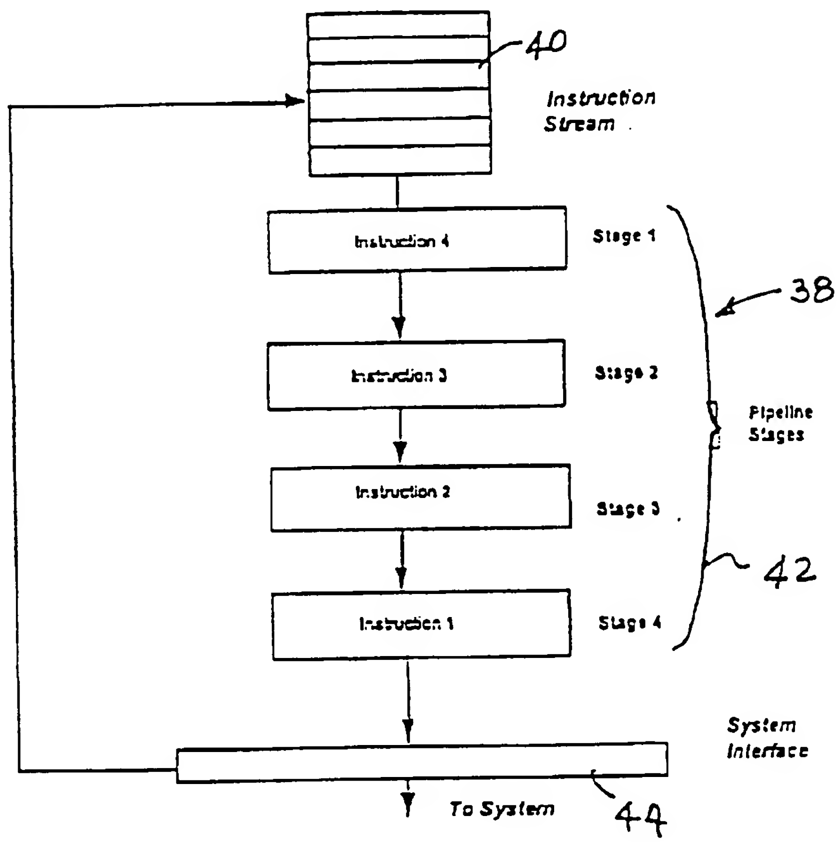

40
Instruction Stream
Instruction 4
Stage 1
38
Instruction 3
Stage 2
Pipeline Stages
Instruction 2
Stage 3
42
Instruction 1
Stage 4
System Interface
To System
44

Fig. 3

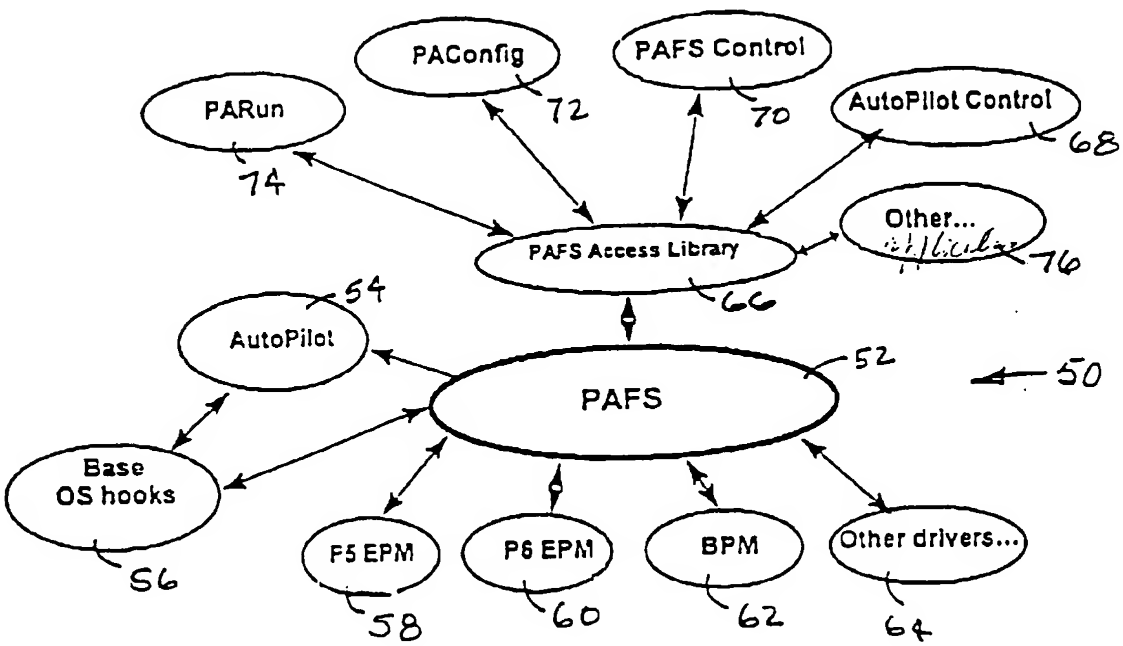

PAConfig
PAFS Control
PARun
AutoPilot Control
Other...
PAFS Access Library
AutoPilot
PAFS
Base OS hooks
F5 EPM
P6 EPM
BPM
Other drivers...
72
70
74
68
76
66
54
52
50
56
58
60
62
64

Fig. 4

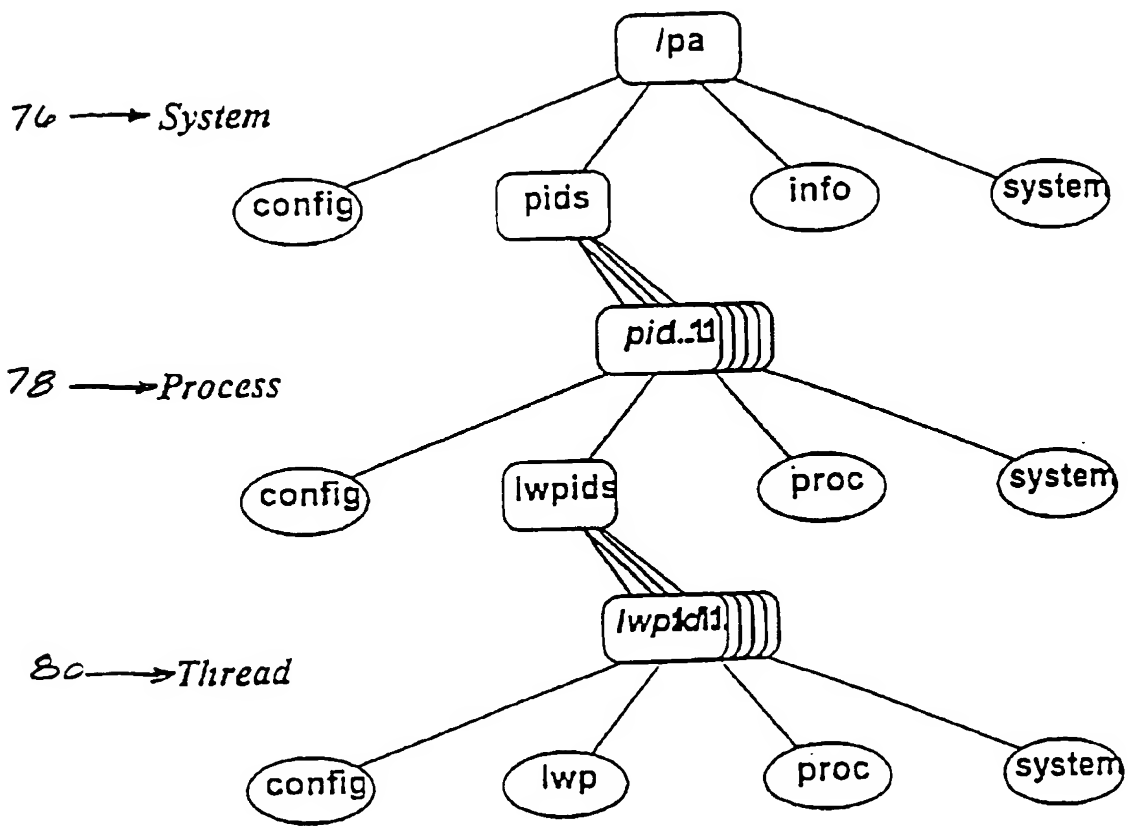

/pa
76 → System
config
pids
info
system
pid..11
78 → Process
config
lwpids
proc
system
80 → Thread
config
lwp
proc
system

Fig. 5